Global Selection Methods for SIMD Computers

Jürgen Branke[1] Hans Christian Andersen[2] Hartmut Schmeck[1]

[1] Institut für Angewandte Informatik und Formale Beschreibungsverfahren
University of Karlsruhe, 76128 Karlsruhe, Germany
E-mail: {branke,schmeck}@aifb.uni-karlsruhe.de

[2] Department of Electrical and Computer Engineering
University of Queensland, St Lucia Qld 4072, Australia
E-mail: andersen@elec.uq.edu.au

Forschungsbericht No. 333, Institute AIFB, University of Karlsruhe, Germany

Abstract

In this paper, the parallel implementation of various selection operators in evolutionary algorithms on SIMD (Single-Instruction-Multiple-Data) computers is treated. Novel parallel versions of fitness proportionate, linear ranking, and tournament selection are presented and compared. It is found that these algorithms can be implemented such that their expected behaviour is identical or very close to those of the sequential algorithms from which they originated, but with the advantage of significant speed improvements associated with parallelism.

Keywords: evolutionary algorithms, genetic algorithms, parallel selection

1 Introduction

Evolutionary algorithms are highly parallel by nature since they concurrently work on a population of candidate solutions. Looking at the typical structure of a genetic algorithm (cf. Fig. 1) it is evident that reproduction, mutation, and especially the time consuming evaluation operation can easily be done in parallel for different individuals on different processors.

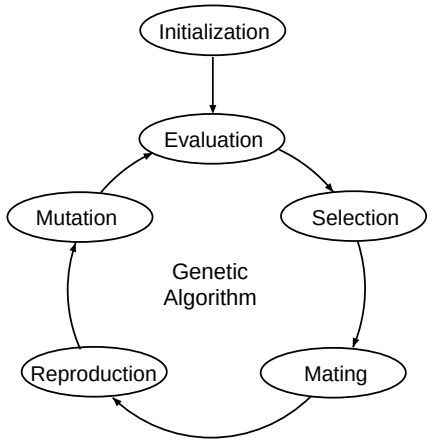

Figure 1: The basic loop of a genetic algorithm

Selection, however, requires global information to determine the relative fitnesses of the individuals. Mating[1] usually involves sending and receiving individuals and thus causes a lot of communication between processors. For evolutionary algorithms without crossover (Evolutionary Programming, Evolution Strategies), the mating is omitted (unary reproduction), but the problem of parallelising selection remains.
To circumvent this problem, researchers usually restrict selection and mating to subsets of the population when designing parallel genetic algorithms, either by

A. introducing subpopulations that work largely independent of each other, except for occasional exchanges of individuals (island model, see e.g. [12]) or

B. defining a spatial distribution on the population and restricting selection and mating to the individual's local neighbourhood (diffusion model, see e.g. [11]).

Approach A is well suited to MIMD-computers where each of the relatively powerful processors can host a whole subpopulation. Approach B is especially suited to SIMD-machines where each processor is assigned a single individual and the spatial layout is defined by the processor interconnectivity.
However, these aforementioned *distributed* algorithms are substantially different from the original sequential algorithm [6, 9] since they renounce global information; they are not *global*[2]. We consider a selection scheme to be global if the selection probability of each individual is based on its fitness relative to the fitness of *all* other individuals in the entire population, regardless of where they are located. In this paper, three *global* selection schemes for SIMD-computers are proposed, examined, and compared.
The outline of the paper is as follows: Section 2 recalls three basic selection schemes used for sequential algorithms. In Section 3, parallel versions of these three selection schemes are presented. A summary and various pertinent conclusions can be found in Section 4.

1.1 Notation

The following notation is used throughout this paper: N refers to the number of individuals in the population; f_i is the fitness of individual i; p_i is the probability that a particular individual is selected in a *single* selection step; $E(i)$ is the expected number of copies of individual i after N selection steps (thus $E(i) = Np_i$); σ_i refers to the prefix sum associated with individual i; and ρ_i is the rank of individual i with respect to fitness, ranging from 1 (least fit) to N (most fit).

2 Sequential Selection

Selection plays an important role in evolutionary computation since it determines which individuals are allowed to survive and to reproduce. Without selection, evolutionary algorithms would be not much different from random walk. Only selection directs the process towards more promising regions of the search space.

[1] By the term mating we mean bringing together two parents for reproduction. On parallel architectures where different individuals may be located on different processors, this may be expensive.

[2] Of course it is widely accepted that such local selection schemes may in some cases have a positive effect on the algorithm since they allow individuals to develop in more or less isolated niches. This issue, however, is not addressed in this paper.

The literature suggests several different selection schemes, the most prominent are probably fitness proportionate selection, linear ranking selection, and tournament selection. In this section we shall briefly recall these selection schemes.
Comparisons of different selection schemes by means of characteristics like selection pressure, loss of diversity, or takeover time can be found in [1, 2, 3, 7, 8]. The main characteristics that we use in this paper to describe and compare the algorithms is the selection probability, p_i, the expected number, $E(i)$, of copies of an individual after the whole selection process, and the required computational complexity.

2.1 Sequential Fitness Proportionate Selection

This method was proposed by Holland in his pioneering work on genetic algorithms [9] and is traditionally the most commonly used selection operator. It yields selection probabilities as follows:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{1}$$

where f_i is the fitness of individual i, and N is the number of individuals in the population.

It can be implemented by Algorithm 1 (usually referred to as the *Roulette Wheel* selection method).

Algorithm 1 Sequential fitness proportionate selection

1.	Calculate *prefix sums*, $\sigma_i = \sum_{j=1}^{i} f_j$, for each individual.	$O(N)$
N times:		
2.	Generate a random number, r, between 0 and σ_N	$O(1)$
3.	Select the individual, k, for which $\sigma_{k-1} < r \leq \sigma_k$ (we define $\sigma_0 = 0$).	$O(\log N)$

2.2 Sequential Linear Ranking Selection

Linear ranking selection involves ranking the individuals with respect to their fitness, and then selecting parents on the basis of their rank rather than their fitness value.
Each individual is assigned a rank, ρ_i, between 1 (worst) and N (best). No two individuals may have the same rank, and cases where the fitnesses of individuals are equal may be resolved arbitrarily. It is common to assign a selection probability $p_1 = 0$ to the worst individual, $p_N = \frac{2}{N}$ to the best individual, and to interpolate linearly in between. This may also be expressed in the following equation:

$$p_i = \frac{\rho_i - 1}{\sum_{j=1}^{N}(j-1)} = \frac{2(\rho_i - 1)}{N(N-1)} \tag{2}$$

It thus follows that the expected number of individuals after N selections is:

$$E_{rank}(i) = \frac{2(\rho_i - 1)}{N-1} \tag{3}$$

To select from a population with this probability the procedure given in Algorithm 2 may be used.

Algorithm 2 Sequential linear ranking selection

1.	Sort individuals wrt fitness.	$O(N \log N)$
N times:		
2.	Generate a random integer, r, such that $0 \leq r < \frac{1}{2}N(N-1)$	$O(1)$
3.	Calculate the *selected rank*, $\rho_r = \left\lfloor \sqrt{\frac{1}{4} + 2r} + \frac{1}{2} \right\rfloor$	$O(1)$
4.	Select the individual with a rank equal to ρ_r.	$O(1)$

Note that since the individuals of the population are sorted in order of increasing fitness before selection commences, the selected rank, ρ_s, can be used as an index into the data structure containing the populations. Fetching the parent can therefore be done without having to search as is required in conventional fitness proportional selection.

2.3 Sequential Tournament Selection

With tournament selection, a single individual is selected by randomly (usually with replacement) choosing t ($t \geq 1$, tournament size) individuals from the population and selecting the best individual out of this group. A common tournament size is $t = 2$, but a generalisation to larger t is possible. See Algorithm 3.

Algorithm 3 Sequential tournament selection

N times:		
1.	Choose t individuals from the population at random with replacement.	$O(t)$
2.	Select the individual with the highest fitness in the tournament group.	$O(t)$

For individual i to be selected in a particular selection step, it must be in the tournament group AND all of the other individuals in the tournament group must have a rank which is less than or equal to ρ_i. The formulas below give the probability of this event to occur.
For tournament selection with replacement, the selection probability for individual i ist:

$$p_i = \frac{\rho_i^t - (\rho_i - 1)^t}{N^t} \tag{4}$$

The expected number of copies of individual i selected in N selection steps is given by:

$$E_{tour}(i) = \frac{\rho_i^t - (\rho_i - 1)^t}{N^{t-1}} \tag{5}$$

Without replacement the probability of selection becomes:

$$p_i = \frac{\binom{\rho_i}{t} - \binom{\rho_i - 1}{t}}{\binom{N}{t}} \tag{6}$$

resp.

$$E_{tour}(i) = N \frac{\binom{\rho_i}{t} - \binom{\rho_i - 1}{t}}{\binom{N}{t}} \tag{7}$$

where again it is assumed that no two individuals share the same rank (cases where the fitnesses of individuals are equal may be resolved arbitrarily).

Note that although this selection method has a partially local character (select best individual out of a small group of individuals), it is still global, since the tournament group is drawn anew from the whole population for every member of the new population.

3 Parallel Selection

As was mentioned in the introduction, the selection (with associated *fetching*[3] operation) as well as mating are those parts of a parallel algorithm which will usually cause the most inter-processor communication. In this section we shall describe parallel variants of each of the selection algorithms of the previous section.
For the discussion of these algorithms we will assume a population of N individuals on N processors connected as a $\sqrt{N} \times \sqrt{N}$ mesh-connected array. In our statements on complexity we assume that standard mesh algorithms are used for sorting, routing, prefix computation etc. (cf. [10]). Since many SIMD computers (like e.g. MasPar) are additionally equipped with a global routing network, we also make some comments on how that network might be used.
For the algorithms below, it is furthermore assumed that the offspring completely replaces the parent generation, as is common in the area of genetic algorithms (see [6]), and that two parent individuals produce two offsprings. However, it is relatively easy to modify the presented algorithms such that e.g. the N processors host $2N$ individuals, that the offspring competes with its parents for survival, or that two parents produce only one child.

3.1 Parallel Fitness Proportionate Selection

The algorithm we propose here performs four basic steps: firstly, the roulette wheel is built, i.e. each processor gets to know which part of the roulette wheel it represents. Then, each processor selects a parent individual by drawing a random number. Thirdly, the random numbers are matched with the roulette wheel segments and the addresses of the processors of the corresponding roulette wheel segment are determined. Finally, the individuals are sent to the processors that requested it. For mating, it is sufficient to just have the processors exchange copies of their parent individuals with their right or left neighbour since these are random mating partners. The algorithm is detailed in Algorithm 4.
The total complexity of this operation is $O(\sqrt{N})$. Given a computer platform where a fetch-operation is available that uses a global routing network (like e.g. on a MasPar), it may be advantageous to stop after step 4, have each processor send π_j to processor j and then have each processor fetch its individual from processor π_j. Doing so might result in a worse worst case complexity (e.g. if all processors request the same individual), but in a better average case complexity (since the gobal routing network should be faster).

3.2 Parallel Linear Ranking Selection

For Linear Ranking Selection, the rank of each individual can be easily obtained by a parallel sorting operation (cf. step 1 in Algorithm 5). Determining the rank of a selected individual is easy as well, since the probabilities for selection do only indirectly depend on the fitness values. Fetching the desired individuals (steps 3 to 9 in Algorithm 5) is not as straightforward, if special cases (e.g. all processors request the same individual) should still be handeled in $O(\sqrt{N})$ time.

[3]The term *fetching* is used to describe the process of retrieving a selected individual from the processor on which it is located.

Algorithm 4 Parallel fitness proportionate selection

1. Calculate prefix sums σ_i with respect to the fitness values and broadcast σ_N (the total fitness) to all processors. $O(\sqrt{N})$
2. In each processor, j, draw a random number r_j between zero and σ_N. $O(1)$

The following steps 3 and 4 serve to determine the address π_j of the selected parent, i.e. if $\sigma_{i-1} < r_j \leq \sigma_i$, then $\pi_j = i$.

3. Merge the two sequences $(\sigma_1, ..., \sigma_N)$ and $(r_1, ..., r_N)$ into snakelike order such that each processor receives two numbers[a]. $O(\sqrt{N})$
4. Count the number of σ_i's smaller than an r_j by another parallel prefix computation (leading to prefix sums $c_1, ..., c_{2N}$). If r_j is the k'th r-element of the sorted sequence (step 3), then $c_k + 1$ is the address of the processor storing the parent selected by processor j, i.e. $\pi_j = c_k + 1$. $O(\sqrt{N})$

By the following steps 5 to 9, the selected parents are sent to their destination processors.

5. Sort all pairs (π_j, j) into snakelike order (wrt the first components). This results in the sequence $(\pi_{j_1}, j_1), (\pi_{j_2}, j_2), ..., (\pi_{j_N}, j_N)$. $O(\sqrt{N})$
6. If $\pi_{j_{k-1}} \neq \pi_{j_k}$ then processor k sends its address (k) to processor π_{j_k}. $O(\sqrt{N})$
7. If processor i received an address k in step 6, it sends its individual, x_i, to processor k. $O(\sqrt{N})$
8. If processor k received an individual x in step 7, it broadcasts (using multicast) x to all the processors k' for which $\pi_{j_k} = \pi_{j_{k'}}$. After this step, every processor contains an individual x and a pair (π_j, j), x having been originally requested by processor j. $O(\sqrt{N})$
9. Each processor sends "its" individual x to processor j. $O(\sqrt{N})$

After this step, every processor has received the individual corresponding to the random number chosen in step 2.

Mating:

10. Each processor: fetch second parent chromosome from neighbouring processor. Odd processors fetch from the left and evens fetch from the right. $O(1)$
11. Perform crossover on parent chromosomes to form offspring. $O(1)$

[a] Actually it is necessary to use pairs (r_j, j) instead of just r_j because the index j (address of selecting processor) is needed later on.

The passing of pointers to the processors where particular chromosomes reside, rather than the passing of the actual chromosomes, is done in order to reduce interprocessor communication.
If a global routing network is available, one might instead have each processor s fetch from processor ρ_s the pointer p_{ρ_s} and then fetch its individual from processor p_{ρ_s}.
Mating, again, is easy since the local neighbours can function as random mating partners.

Algorithm 5 Parallel linear ranking selection

Selection:

1.	Let f_i be the fitness of the individual on processor i. Sort pairs (f_i, i) wrt. their first components. As a result, each processor j contains a pair (f_i, i) such that individual i has rank j, i.e. processor j has a pointer, $p_j = i$, to the processor on which the individual with rank j is located	$O(\sqrt{N})$
2.	Each processor: generate a random number, r_s, between 0 and $\frac{1}{2}N(N-1)-1$ (inclusive), and calculate the selected rank, $\rho_s = \left\lfloor \sqrt{\frac{1}{4} + 2r_s} + \frac{1}{2} \right\rfloor$.	$O(1)$
3.	Sort the selected ranks ρ_s into snakelike order. This results in a sequence $(\rho_{s_1}, \rho_{s_2}, ..., \rho_{s_N})$	$O(\sqrt{N})$
4.	If $\rho_{j_{k-1}} \neq \rho_{j_k}$ then processor k sends its address (k) to processor ρ_{j_k}.	$O(\sqrt{N})$
5.	If processor l received an address k in step 4, it sends its pointer p_l back to processor k.	$O(\sqrt{N})$
6.	If processor k received a pointer p_l in step 5, it sends its address (k) to processor p_l.	$O(\sqrt{N})$
7.	If processor i received an address k in step 6, it sends its individual, x_i, to processor k.	$O(\sqrt{N})$
8.	If processor k received an individual x in step 7, it broadcasts (using multicast) x to all the processors k' for which $\rho_{j_k} = \rho_{j_{k'}}$. After this step, every processor contains an individual x and a pair (ρ_j, j), x having been originally requested by processor j.	$O(\sqrt{N})$
9.	Each processor sends "its" individual x to processor j.	$O(\sqrt{N})$

Mating:

10.	Each processor: fetch second parent chromosome from neighbouring processor. Odd processors fetch from the left and evens fetch from the right.	$O(1)$
11.	Perform crossover on parent chromosomes to form offspring.	$O(1)$

3.3 Parallel Spatial Tournament Selection

In this section, we propose a parallel variant of the tournament selection method where the tournaments are laid out spatially, i.e. tournaments are held in the local neighbourhood of each processor. We therefore call this method Spatial-Tournament (ST) Selection. As we will show, it exhibits very similar expected behaviour to the linear ranking or sequential tournament methods, but is much easier to parallelise. The selection method is global since it combines local tournament selection with mixing the whole population spatially.
The exact procedure is given in Algorithm 6.

The local neighbourhood is defined by the processor connectivity and usually will encompass $n = 2, 3$ or 4 other processors. Note that the mixing operation for mating (step 3) may be unnecessary. It is not possible, as in the other proposed algorithms, to just mate with one's right or left neighbour, since this neighbourhood has already influenced the selection process

Algorithm 6 Parallel spatial tournament selection

Selection:

1.	mix individuals	
	- each processor: choose random number	$O(1)$
	- sort individuals by random number	$O(\sqrt{N})$
2.	each processor: select best individual in neighbourhood	$O(\mathrm{n})$

Mating:

(3).	mix individuals	
	- each processor: choose random number	$O(1)$
	- sort individuals by random number	$O(\sqrt{N})$
4.	Each processor: fetch second parent chromosome from neighbouring processor. Processors in odd rows fetch from the north and processors in even rows fetch from the south.	$O(1)$
5.	Perform crossover on parent chromosomes to form offspring.	$O(1)$

and it would no longer be a random mating. However, it would be sufficient to ensure that the mating partners' neighbourhoods do not overlap. For example, for neighbourhood size of two (one's right neighbour) it would be acceptable if all individuals in even rows would mate with their southern neighbours, all individuals in odd rows with their neighbours to the north. This speeds up the algorithm by a factor of two, approximately.

Theorem 1 *The expected number of instances of an individual with rank ρ_i after ST-Selection with neighbourhood-size 2 is*

$$E_{ST,n=2}(i) = \frac{2(\rho_i - 1)}{N - 1}. \tag{8}$$

Proof: W.l.o.g. we can assume that individual i is located on processor x as shown in Figure 2. Then there are 2 ways in which it can be selected: (1) by processor x itself if it's fitness is greater than that of the individual's right-hand neighbour, or (2) by the processor to it's left if the fitness of the individual on that processor is lower.
The probability of each of these is $\frac{\rho_i - 1}{N-1}$, and so the total probability of individual i being selected is $2\frac{\rho_i - 1}{N-1}$. ■

Theorem 2 *Given a neighbourhood-size of n, the expected number of instances of an individual of rank ρ_i after ST-selection is*

$$E_{ST}(i) = n \prod_{\nu=1}^{n-1} \frac{\rho_i - \nu}{N - \nu} \tag{9}$$

.

Proof: Again, w.l.o.g. one can assume that the individual with rank ρ_i has been placed on processor x. With neighbourhood size n, individual i belongs to the neighbourhood of n processors. From each of these n processors it is selected if the processor's $(n-1)$ other neighbours all have

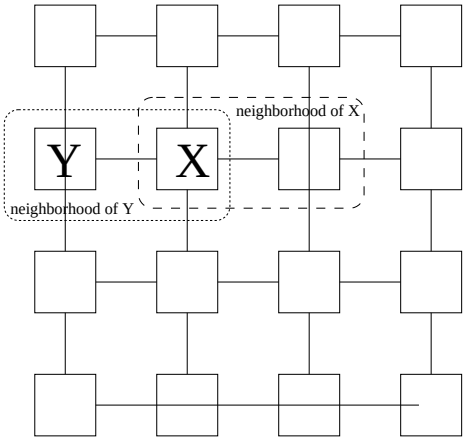

Figure 2: 2-neighbourhoods encompassing processor x.

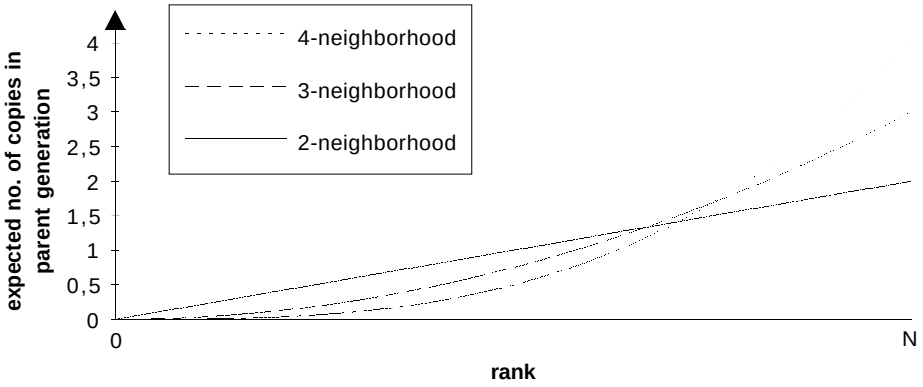

Figure 3: The expected value of of an individual's copies after ST-selection with neighbourhood $n = 2, 3$ and 4.

rank less than ρ_i. There are exactly $(\rho_i - 1)$ individuals with lower rank and the probability for all $n - 1$ neighbours of one processor having a lower rank is

$$\frac{\binom{\rho_i - 1}{n-1}}{\binom{N-1}{n-1}} = \prod_{\nu=1}^{n-1} \frac{\rho_i - \nu}{N - \nu}.$$

■

Some example graphs of $E(i)$ for neighbourhood sizes $n = 2, 3$ and 4 are displayed in Figure 3.

Theorem 3 *The expected number of copies of an individual after ranking selection, tournament selection with $t = 2$ and no replacement, and ST-selection with $n = 2$ are identical, i.e.*

$$E_{rank}(i) = E_{tour,t=2}(i) = E_{ST,n=2}(i); i = 1, 2, \ldots N \tag{10}$$

Proof: Equations 3 and 8 are identical for $n = t = 2$. Substituting $t = 2$ into equation 7 and breaking up the a-choose-b expressions yields again the same equation as equation 8. ■

But while having the same reproduction rates certainly makes these three schemes very similar, they are not equal:

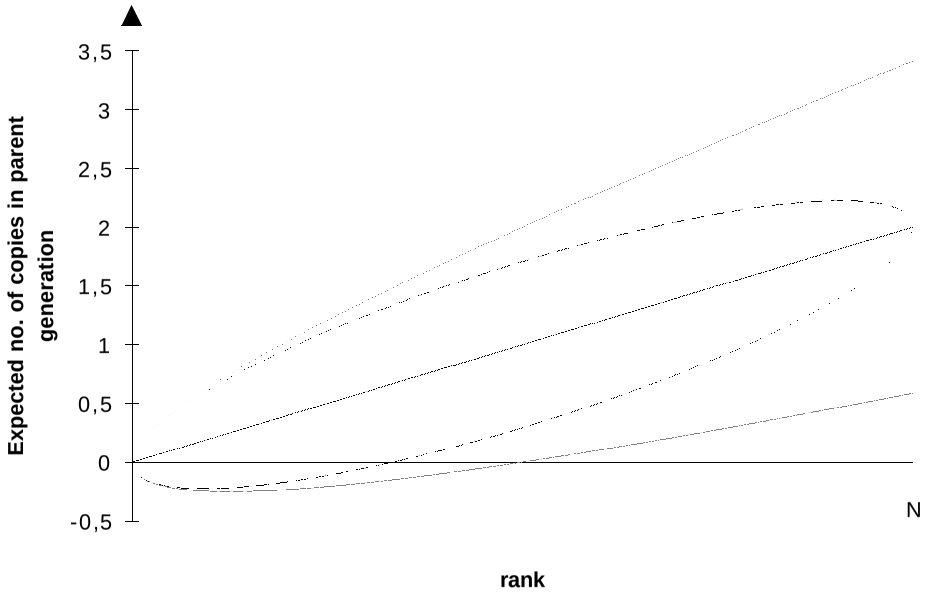

Figure 4: Expected number of offspring $\pm$ std. dev. for ranking selection (outer curves) resp. std. dev. for ST-selection (inner curves).

- While with rank based selection, except for the least fit individual, the actual number of copies from an individual after selection may vary between 0 and N, with ST-selection the maximum number of copies from one individual to the next generation is n.
- ST-selection has a different variance and standard deviation for the expected number of copies of the individuals. For example with ST-selection, the best individual is chosen exactly twice, while with rank based selection there is much more variance. Figure 4 shows the curves of the expected reproduction values plus and minus the standard deviation for both selection schemes.

The effect of this reduced variance is yet unclear. One might speculate, however, that the reduced variance leads to more consistent results over different runs. The importance of selection variance has recently been shown by De Jong and Sarma [4] who observed better search performance was reached with smaller selection variance, which would support our expectation.
The fact that the best individual is chosen exactly n times makes ST-selection an elitist selection strategy: the best individual is never lost. With most other selection schemes, this feature has to be enforced by extra efforts.

3.4 Extensions to ST-Selection

There are several extensions to the simple ST-selection scheme possible:

- With almost no extra computational effort, one might have each processor choose its own individual independent from the neighbourhood with some predefined probability ϵ, otherwise proceed as usual. This modification flattens out the curve of expected copies after selection and at the same time introduces some additional variance. The resulting $E^*(i)$ is just a linear combination of the old $E(i)$ and $f(i) = 1$, i.e.

$$E^*_{ST}(i) = \epsilon + (1 - \epsilon)E(i) \tag{11}$$

- another extension would be to use fitness proportionate or ranking selection instead of tournament selection in the local neighbourhoods.

4 Conclusions

When parallelising evolutionary algorithms, the selection and mating operations are the major bottleneck.
In this paper, we presented three fast implementations of global selection algorithms for massively parallel SIMD (Single Instruction Multiple Data) computer architectures.
The first two parallel selection schemes were just efficient parallel versions of standard fitness proportionate selection resp. linear ranking selection.
The third algorithm presented, called Spatial Tournament (ST)-Selection combines the partially local character of tournament selection with a global random assignment of individuals to tournament groups. As such, it es extremely simple and easy to implement, but exhibits the same expected behaviour as linear ranking selection or tournament selection without replacement. The selection variance, however, is smaller, and since the best individual is chosen exactly twice, it implicitly is an elitist selection strategy.
All the presented algorithms have running time of $O(\sqrt{N})$, compared to $O(N \log N)$ resp. $O(N)$ in the sequential case.
So, while nowadays most evolutionary algorithms on massively parallel SIMD machines are variants of the diffusion model, the algorithms presented in this paper may encourage to alternatively implement parallel versions of the ordinary global evolutionary algorithm or the island model (as has been done by [5]) on the SIMD computer platform.
Future research in this area will include empirical evaluations on the proposed algorithms, the design of more hardware-customized selection schemes like ST-selection and a closer investigation of the influence of selection variance on the run of the evolutionary algorithm.

References

[1] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *1st IEEE conference on evolutionary computing*, volume 1, pages 57–62. IEEE, 1994.

[2] T. Blickle and L. Thiele. A comparison of selection schemes used in genetic algorithms. Technical report, Swiss Federal Institute of Technology (ETH), 1995.

[3] R.J. Collins and D.R. Jefferson. Selection in massively parallel genetic algorithms. In R.K. Belew and L.B. Booker, editors, *4th International Conference on Genetic Algorithms*, pages 249–256, San Diego, CA, 1991. Morgan Kaufmann.

[4] K. De Jong and J. Sarma. On decentralizing selection algorithms. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms, ICGA'95*, pages 17–23. Morgan Kaufmann Publishers, San Francisco, July 1995.

[5] D. Eichberg, U. Kohlmorgen, and H. Schmeck. Feinkörnig parallele Varianten des Insel-Modells Genetischer Algorithmen. In *GI-Mitteilungen: Workshop für Parallel-Algorithmen und Rechnerstrukturen*. Gesellschaft für Informatik e. V., 1995.

[6] D. E. Goldberg. *Genetic Algorithms*. Addison-Wesley, 1989.

[7] D. E. Goldberg and D. Kalayanmoy. A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, San Mateo, CA, USA, 1991. Morgan Kaufmann.

[8] P.J.B. Hanrock. An empirical comparison of selection methods in evolutionary algorithms. In *AISB workshop on evolutionary computation*, 1994.

[9] J. H. Holland. *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press, 1975.

[10] F. T. Leighton. *Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.

[11] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337, San Mateo, CA, 1991. Morgan Kaufmann.

[12] D. Whitley and T. Starkweather. Genitor ii: a distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(3):189–214, 1990.